Summary of Basics about Generative Adversarial Network

Presentor: Ji Gao

Department of Computer Science, University of Virginia
https://qdata.github.io/deep2Read/
Generative model

• Training and sampling from generative model shows how our model can represent high-dimensional probability distribution
• Generative model can be used on reinforcement learning
• Generative model can perform semi-supervised learning
• Generative model can enable machine learning to work with multi-modal output
• Many tasks intrinsically require the generation of good samples
Maximum Likelihood Estimation

• Maximum likelihood:
  • Choose the parameters to maximize the likelihood to training data
  • Easier to do in the log space

\[
\theta^* = \arg \max_{\theta} \prod_{i=1}^{m} p_{\text{model}}(x^{(i)}; \theta) \\
= \arg \max_{\theta} \sum_{i=1}^{m} \log p_{\text{model}}(x^{(i)}; \theta) \\
= \arg \max_{\theta} \int p_{\text{data}}(x) \log \frac{p_{\text{model}}(x; \theta)}{p_{\text{model}}(x)} \, dx \\
= \arg \min_{\theta} \int p_{\text{data}}(x) \log \frac{p_{\text{data}}(x)}{p_{\text{model}}(x)} \, dx \\
= \arg \min_{\theta} D_{KL}(p_{\text{data}} \| p_{\text{model}}(\theta)) \quad \text{KL divergence!}
\]
Generative model Tree (NIPS 2016 Tutorial)
Autoregressive models

$$p_{\text{model}}(x; \theta) = \prod_{d=1}^{D} p(x_d | x_{1:d-1}; \theta)$$

Deep Belief Network
NADE
MADE
PixelRNN
PixelCNN++
WaveNet

- Define explicit density function
- Using chain rule
Variational Autoencoder

• Maximize variational lower bound $L(x; \theta) \leq \log p_{model}(x; \theta)$

• Guarantee to achieve a high value as the log-likelihood

• Issue:
  • Bias: If not properly defined, the gap might be an issue
  • Tend to get good likelihood, but bad quality on image
Generative Adversarial Network (GAN)

- **Generator**: Some differentiable function $G(z)$. $Z$ is sampled from some simple prior distribution, $G(z)$ is a sample of $x$ drawn from $p_{model}$.
  - Structure can be anything
- **Discriminator**: Traditional supervise learning model
Math: Cost Function of GAN

• **Discriminator:** $J^{(D)}(\theta^{(D)}, \theta^{(G)}) =
- \mathbb{E}_{x \sim p_{data}} \log D(x) - \mathbb{E}_{z} \log(1 - D(G(z)))$

• Cross entropy loss. Label 1 for the samples in the dataset, 0 for generator samples.

• Discriminator is able to estimate $p_{data}(x)/p_{model}(x)$ everywhere

• It doesn’t have any bias.

• **Generator:** $J^{(G)}(\theta^{(D)}, \theta^{(G)}) = -J^{(D)}(\theta^{(D)}, \theta^{(G)})$ if it’s a zero-sum game

• In zero-sum game case the game resembles Jenson-Shannon Bound.
Jenson-Shannon Bound

• Suppose D is optimal, we have $D(x; \theta_D^*) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_{\text{model}}(x)}$

• In this case $J(\theta_D^*, \theta_G) = E_{x \sim p_d} \log D(x) + E_z \log \left(1 - D\left(G(z)\right)\right)$

• $= \int p_{\text{data}}(x) \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_{\text{model}}(x)} \, dx + \int p_g(x) \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_{\text{model}}(x)} \, dx$

• $= KL\left(p_{\text{data}} \parallel \frac{p_{\text{data}}(x) + p_{\text{model}}(x)}{2}\right) + KL\left(p_{\text{model}} \parallel \frac{p_{\text{data}}(x) + p_{\text{model}}(x)}{2}\right) - 2 \log 2$

• $= 2Jenson–Shannon(p_{\text{data}} \parallel p_{\text{model}}) - 2 \log 2$
Math: Cost Function of GAN II

• **Discriminator:** $J^{(D)}(\theta^{(D)}, \theta^{(G)}) = -E_{x \sim \rho_{data}} \log D(x) - E_{z} \log(1 - D(G(z)))$

• **Generator:** $J^{(G)}(\theta^{(D)}, \theta^{(G)}) = -J^{(D)}(\theta^{(D)}, \theta^{(G)})$ if it’s a zero-sum game

• However, zero-sum game doesn’t perform well in practice:
  • Generator maximize the cross entropy
  • If discriminator performs well, the gradient of generator vanishes

• Solution from Goodfellow 2014: Change the sign
  $J^{(G)}(\theta^{(D)}, \theta^{(G)}) = -E_{z} \log(D(G(z)))$

• No longer zero-sum, and heuristically designed
Math: Cost Function of GAN III

- If let $J^G(\theta^D, \theta^G) = -E_{Z} \sigma^{-1} \left( D(G(Z)) \right)$:
- $J^G(\theta^D, \theta^G) = \int p_{model}(x) e^{\sigma^{-1}(D(G(z)))} dx$
- We have $e^{\sigma^{-1}(x)} = \frac{x}{1-x'}$, therefore $e^{\sigma^{-1}(D(x)^*)} = \frac{p_{data}(x)}{p_{model}(x)}$
- $J^G(\theta^D, \theta^G) = - \int p_{model}(x) \log \frac{p_{data}(x)}{p_{model}(x)} dx = D_{KL}(p_{model} \parallel p_{data})$

- By changing loss function from g, GAN can work on different loss
Choosing loss functions

- VAE use KL(MLE), which tends to get blurred image
- Reverse KL on the other side, tend to converge to modes
- GAN(original version) optimize JS, somehow similar to reverse KL
- However, it sometimes generate samples only from few modes
Choosing loss functions

• If the optimization is easy, minimax give no gradient

• The change is rapid on the right side, makes small number of samples dominating
Architecture: DCGAN

• Keys:
  • Use batch normalization, with the two minibatches for the discriminator normalized separately
  • All-convolutional net (Springenberg et al., 2015), no pooling
  • Adam instead of SGD
GAN tricks

• [https://github.com/soumith/gansocks](https://github.com/soumith/gansocks)
• Train with labels can improve the performance drastically
• “One-sided” Label smoothing: Replace the target for the real examples with a value slightly less than 1, such as .9
• Batch normalization
• Adam
Convergence of GAN(Or Nonconvergence)

• For two player game, even if each player successfully moves downhill on that player’s update, the same update might move the other player uphill.
• It converge on some game but not all.
• Mode collapse could be one result of such nonconvergence (rather than loss function). WGAN claims it alleviate the mode collapse problem, though.
Non-Convergence of GAN: Toy Example

• Toy example of game:
  • $V(x, y) = xy$
  • P1 Minimize $V(x, y)$ by controlling $y$
  • P2 Maximize $V(x, y)$ by controlling $x$
• Equilibrium: $x = y = 0$
• Gradient: $\Delta x = -\alpha \frac{\partial V}{\partial x} = -\alpha y$
  • $\frac{\partial y}{\partial t} = \alpha \frac{\partial V}{\partial y} = \alpha x$
Convergence proof

• If $g$ is on all functions, $p_{model}$ converge to $p_{data}$

• However, it’s not the case in the deep neural network

Proposition 2. If $G$ and $D$ have enough capacity, and at each step of Algorithm 1, the discriminator is allowed to reach its optimum given $G$, and $p_g$ is updated so as to improve the criterion

$$\mathbb{E}_{x \sim p_{data}}[\log D_G^*(x)] + \mathbb{E}_{x \sim p_g}[\log(1 - D_G^*(x))]$$

then $p_g$ converges to $p_{data}$

Proof. Consider $V(G, D) = U(p_g, D)$ as a function of $p_g$ as done in the above criterion. Note that $U(p_g, D)$ is convex in $p_g$. The subderivatives of a supremum of convex functions include the derivative of the function at the point where the maximum is attained. In other words, if $f(x) = \sup_{\alpha \in \mathcal{A}} f_\alpha(x)$ and $f_\alpha(x)$ is convex in $x$ for every $\alpha$, then $\partial f_\beta(x) \in \partial f$ if $\beta = \arg \sup_{\alpha \in \mathcal{A}} f_\alpha(x)$. This is equivalent to computing a gradient descent update for $p_g$ at the optimal $D$ given the corresponding $G$. $\sup_D U(p_g, D)$ is convex in $p_g$ with a unique global optima as proven in Thm 1, therefore with sufficiently small updates of $p_g$, $p_g$ converges to $p_x$, concluding the proof. \qed
Evaluation of generative models

- Hard to evaluate because doesn’t give probability on the training data
- Inception score?

Other scores:
- MODE score
- AM Score
Reference

• 2. https://github.com/soumith/ganhacks